CENTRUL DE PREGĂTIRE PENTRU PERFORMANŢĂ HAI LA OLIMPIADĂ !
DISCIPLINA **INFORMATICĂ**
JUDEŢUL **SUCEAVA**

Titlul lectiei: Algoritmul lui Lee
Data: 26.11.2016
Profesor: Maghiuc Florin
Grupa: clasa a X-a **locul de desfasurare**: CN PR Suceava

Algoritmul lui Lee

Introducere

În continuare vom prezenta *algoritmul lui Lee*, pentru cei care nu ştiu, este identic cu *parcurgerea în lățime* doar ca e aplicat pe o grila, nu pe un graf oarecare. Este eficient, având o complexitate de O(M*N), şi frecvent utilizat. Acesta determină drumul minim de ieşire dintr-un labirint, sau în probleme asemănătoare.

Prezentare

Algoritmul lui Lee presupune doi paşi importanți:

1. Primul şi poate cel mai important pas este folosirea unei **Cozi**, sub forma unui vector de structuri (de preferabil), care va menține toți paşii pe care o să-i facem de acum în colo. În această coadă se pun, pentru fiecare pas, locurile care s-au marcat la punctul anterior.
2. Se marchează cu numere consecutive toate locurile posibile prin care putem trece, parcurgând în ordine elementele cozii, până când nu mai putem marca, sau am ajuns la final.

Implementare

```
#include <bits/stdc++.h>

using namespace std;
```

```
ifstream fin("lee.in");
ofstream fout("lee.out");

const int NMax = 1005;
const int dx[] = {0, -1, 0, 1};
const int dy[] = {-1, 0, 1, 0};

int v[NMax][NMax];

void lee(int x, int y) {
   deque < int > qx, qy;
   qx.push_back(x);
   qy.push_back(y);

   v[x][y] = 1; //initializam pozitia de pornire pentru a nu trece prin ea de doua ori
   while(qx.empty() == false) { //cat timp mai avem elemente de "prelucrat"
      x = qx.front(); //scoatem coordonatele elementelor
      y = qy.front(); //din coada
      qx.pop_front();
      qy.pop_front();

      for(int i = 0; i < 4; i++) { //alegem toti vecinii elementului respectiv
         int nx = x + dx[i];
         int ny = y + dy[i];

         if(v[nx][ny] == 0) { //verificam daca am trecut sau nu prin acest vecin
            v[nx][ny] = v[x][y] + 1;
            qx.push_back(nx);
            qy.push_back(ny);
         }
      }
   }
}

int main() {

   int n, m; //dimensiunile matricii
   fin >> n >> m;

   int a, b; //pozitiile de pornire
   fin >> a >> b;

   int c, d; //pozitiile de oprire
   fin >> c >> d;
```

```
    int k;
    fin >> k; //numarul de obstacole

    for(int i = 1; i <= k; i++) {
        int x, y;
        fin >> x >> y;

        v[x][y] = -1; //notam in matrice obstacolele cu -1
    }

    for(int i = 0; i <= m + 1; i++) v[0][i] = -1; //bordam matricea
    for(int i = 0; i <= m + 1; i++) v[n + 1][i] = -1;
    for(int i = 0; i <= m + 1; i++) v[i][0] = -1;
    for(int i = 0; i <= m + 1; i++) v[i][m + 1] = -1;

    lee(a, b);

    fout << v[c][d]; //afisam distanta de la punctul de pornire la cel de oprire
}
```

Probleme

Alee, Muzeu, Insule, Tsunami, Figuri

Problema 1 – Alee 100 puncte

Parcul oraşului a fost neglijat mult timp, astfel că acum toate aleile sunt distruse. Prin urmare, anul acesta Primăria şi-a propus să facă reamenajări.
Parcul are forma unui pătrat cu latura de n metri şi este înconjurat de un gard care are exact două porţi. Proiectanţii de la Primărie au realizat o hartă a parcului şi au trasat pe hartă un caroiaj care împarte parcul în nxn zone pătrate cu latura de 1 metru. Astfel harta parcului are aspectul unei matrice pătratice cu n linii şi n coloane. Liniile şi respectiv coloanele sunt numerotate de la 1 la n. Elementele matricei corespund zonelor pătrate de latură 1 metru. O astfel de zonă poate să conţină un copac sau este liberă.
Edilii oraşului doresc să paveze cu un număr minim de dale pătrate cu latura de 1 metru zonele libere (fără copaci) ale parcului, astfel încât să se obţină o alee continuă de la o poartă la alta.

Cerinţă

Scrieţi un program care să determine numărul minim de dale necesare pentru construirea unei alei continue de la o poartă la cealaltă.

Date de intrare
Fişierul de intrare alee.in conține pe prima linie două valori naturale n şi m separate printr-un spațiu, reprezentând dimensiunea parcului, respectiv numărul de copaci care se găsesc în parc. Fiecare dintre următoarele m linii conține câte două numere naturale x şi y separate printr-un spațiu, reprezentând pozițiile copacilor în parc (x reprezintă linia, iar y reprezintă coloana zonei în care se află copacul). Ultima linie a fişierului conține patru numere naturale x1 y1 x2 y2, separate prin câte un spațiu, reprezentând pozițiile celor două porți (x1, y1 reprezintă linia şi respectiv coloana zonei ce conține prima poartă, iar x2, y2 reprezintă linia şi respectiv coloana zonei ce conține cea de a doua poartă).

Date de ieşire
Fişierul de ieşire alee.out va conține o singură linie pe care va fi scris un număr natural care reprezintă numărul minim de dale necesare pentru construirea aleii.

Restricții
- $1 \leq n \leq 175$
- $1 \leq m < n*n$
- Aleea este continuă dacă oricare două plăci consecutive au o latură comună.
- Aleea începe cu zona unde se găseşte prima poartă şi se termină cu zona unde se găseşte cea de a doua poartă.
- Pozițiile porților sunt distincte şi corespund unor zone libere.
- Pentru datele de test există întotdeauna soluție.

Exemplu

alee.in	**alee.out**	**Explicație**
8 6 2 7 3 3 4 6 5 4 7 3 7 5 1 1 8 8	15	O modalitate de a construi aleea cu număr minim de dale este: OOO----- --OO--x- --xO---- ---OOx-- ---xO--- ----OO-- --x-xOO- ------OO (cu X am marcat copacii, cu - zonele libere, iar cu O dalele aleii).

Timp maxim de execuție/test: 1 secundă

Implementare Manghiuc Florin

```
#include <iostream>
#include <fstream>
#include <deque>
```

```
using namespace std;

ifstream f("alee.in");
ofstream g("alee.out");

int v[180][180];
const int dx[] = {-1, 0, 1, 0};
const int dy[] = {0, 1, 0, -1};
deque < int > qx,qy;

void lee()
{
   int nx,ny,yy,xx;
   v[qx.front()][qy.front()] = 1;
   while(!qx.empty()){
      nx = qx.front();
      ny = qy.front();
      for(int i = 0; i <= 3; i++){
         xx = nx + dx[i];
         yy = ny + dy[i];
         if(v[xx][yy] == 0){
            qx.push_back(xx);
            qy.push_back(yy);
            v[xx][yy] = v[nx][ny] + 1;
         }
      }
      qx.pop_front();
      qy.pop_front();
   }
}

int main()
{
   int n,m,xi,yi,xf,yf;
   f >> n >> m;
   for(int i = 1; i <= m; i++){
      f >> xi >> yi;
      v[xi][yi] = -1;
   }
   f >> xi >> yi >> xf >> yf;
   f.close();
   qx.push_back(xi);
   qy.push_back(yi);
   for(int i = 0; i <= n + 1; i++)
      v[0][i] = -1,v[i][n+1] = -1,v[n+1][i] = -1,v[i][0] = -1;
```

```
    lee();
    g << v[xf][yf];
    return 0;
}
```

Implementare Aga Petronela Marinela

```
#include <fstream>
#include<iostream>
using namespace std;
const int dx[]={-7,0,1,0,-1};
const int dy[]={-7,1,0,-1,0};
int a[180][180],n,px1,py1,px2,py2;
void citire()
{
    int i,c,l,p;
    ifstream f("alee.in");
    f>>n>>c;
 for(i=1;i<=c;i++)
 {
     f>>l>>p;
     a[l][p]=-1;
 }
 f>>px1>>py1>>px2>>py2;

    for(i=0;i<=n+1;i++)
        a[i][0]=a[i][n+1]=a[0][i]=a[n+1][i]=-1;



}

void leerec(int x,int y,int pas)
{
    int i,xx,yy;
    a[x][y]=pas;
    for(i=1;i<=4;i++)
    {
        xx=x+dx[i];
        yy=y+dy[i];
        if(a[xx][yy]==0||a[xx][yy]>pas+1)
            leerec(xx,yy,pas+1);
    }
}
```

```
void lee()
{
   int qx[92600],qy[92600],inc,sf,xx,yy,nx,ny,i;
   inc=1;sf=1;
   qx[inc]=px1;
   qy[inc]=py1;
   a[px1][py1]=1;
   while(inc<=sf)
   {
      nx=qx[inc];
      ny=qy[inc];
      for(i=1;i<=4;i++)
      {
         xx=nx+dx[i];
         yy=ny+dy[i];
         if(a[xx][yy]==0)
         {
            sf++;
            qx[sf]=xx;
            qy[sf]=yy;
            a[xx][yy]=a[nx][ny]+1;
         }
      }
      inc++;
   }
}
int main()
{
    ofstream g("alee.out");
   citire();
   int i,j;
  /* for(i=1;i<=n;i++)
   {
      for(j=1;j<=m;j++)
         g<<a[i][j]<<" ";
      g<<endl;
   }*/
  // leerec(px1,py1,1);
   /* for(i=1;i<=n;i++)
   {
      for(j=1;j<=n;j++)
         g<<a[i][j]<<" ";
      g<<endl;
   }
g<<x2<<" "<<y2<<endl;*/
```

```
lee();
    g<<a[px2][py2];
    return 0;
}
```

Problema 1 - Insule **100 puncte**

Arhipelagul *RGB* este format din insule care aparţin ţărilor *R, G* şi *B*. Putem reprezenta harta arhipelagului ca o matrice cu n linii şi m coloane cu elemente din mulţimea {0, 1, 2, 3}. Un element egal cu 0 reprezintă o zonă acoperită de apă; un element egal cu 1 reprezintă o zonă de pământ aparţinând unei insule din ţara *R*, iar un element egal cu 2 reprezintă o zonă de pământ aparţinând unei insule din ţara *G*, iar un element egal cu 3 reprezintă o zonă de pământ aparţinând unei insule din ţara *B*.

Se consideră că două elemente ale matricei sunt *vecine* dacă ele au aceeaşi valoare şi fie sunt consecutive pe linie, fie sunt consecutive pe coloană. Două elemente aparţin aceleiaşi insule dacă ele sunt vecine sau dacă se poate ajunge de la un element la celălalt pe un drum de-a lungul căruia oricare două elemente consecutive sunt vecine.

Pentru a încuraja relaţiile de colaborare dintre ţările *R* şi *G*, se doreşte construirea unui pod care să unească o insulă aparţinând ţării *R* de o insulă aparţinând ţării *G*. Podul trebuie să respecte următoarele condiţii:

- să înceapă pe o zonă cu apă consecutivă pe linie sau coloană cu o zonă aparţinând ţării *R*;
- să se termine pe o zonă cu apă consecutivă pe linie sau coloană cu o zonă aparţinând ţării *G*;
- să traverseze numai zone acoperite cu apă;
- oricare două elemente consecutive ale podului trebuie să fie vecine;
- lungimea podului să fie minimă (lungimea podului este egală cu numărul de elemente traversate de pod).

Cerinţă

Dată fiind harta arhipelagului să se determine câte insule aparţin fiecărei ţări, precum şi lungimea minimă a unui pod care să satisfacă condiţiile din enunt.

Date de intrare

Fişierul de intrare insule.in conţine pe prima linie numerele naturale n şi m, separate prin spaţiu. Pe următoarele n linii este descrisă harta arhipelagului. Pe fiecare dintre aceste n linii sunt scrise câte m valori din mulţimea {0, 1, 2, 3}; valorile nu sunt separate prin spaţii.

Date de ieşire

Fişierul de ieşire insule.out va conţine o singură linie pe care vor fi scrise patru numere naturale separate prin spaţii NR NG NB Lg, unde NR reprezintă numărul de insule aparţinând ţării *R*, NG numărul de insule aparţinând ţării *G*, NB numărul de insule aparţinând ţării *B*, iar Lg lungimea minimă a podului.

Restricţii şi precizări

$1 < n, m \leq 100$

Se garantează că pe hartă există cel puţin un element 1, un element 2 şi un element 0.

Se acordă 40% din punctaj pentru determinarea corectă a numărului de insule din fiecare ţară; se acordă punctaj integral pentru rezolvarea corectă a tuturor cerinţelor.

Începutul şi sfârşitul podului pot să coincidă.
Pentru datele de test există întotdeauna soluție.

Exemplu

insule.in	insule.out	*Explicație*
6 7 1000320 0110313 3333000 2033000 2203011 2000010	4 2 3 4	Țara *R* are 4 insule, țara *G* are 2 insule, iar țara *B* are 3 insule. Lungimea minimă a unui pod care poate fi construit este 4; de exemplu, podul traversează celulele (6,5), (6,4), (6,3), (6,2).

Timp maxim de execuție/test: 1 secundă

```
#include <fstream>
#define MAX 101

using namespace std;
ofstream g("insule.out");
const int dx[]={7,-1,0,1,0};
const int dy[]={7,0,1,0,-1};
int a[MAX][MAX],ni[4],n,m;
void citire()
{
   int i,j;
   char s[102];
   ifstream f("insule.in");
   f>>n>>m;
   for(i=0;i<=m+1;i++)
      a[0][i]=a[n+1][i]=-10;
   for(i=0;i<=n+1;i++)
      a[i][0]=a[i][m+1]=-10;
   f.getline(s,3,'\n');
   for(i=1;i<=n;i++)
   {
      f.getline(s,100,'\n');
      for(j=0;j<m;j++)
         a[i][j+1]=s[j]-'0';
   }
}
void afisare()
{
   int i,j;
   for(i=1;i<=n;i++)
```

```
    {
        for(j=1;j<=m;j++)
        g<<a[i][j]<<" ";
        g<<endl;
    }
}
void fill(int x,int y,int c)
{
    int xx,yy,i;
    a[x][y]=-c;
    for(i=1;i<=4;i++)
    {
        xx=x+dx[i];
        yy=y+dy[i];
        if(a[xx][yy]==c)
            fill(xx,yy,c);
    }
}
void numara()
{
    int i,j;
    for(i=1;i<=n;i++)
        for(j=1;j<=m;j++)
        if(a[i][j]>0)
    {
        ni[a[i][j]]++;
        fill(i,j,a[i][j]);
    }
}
void lee(int x,int y)
{
    int qx[MAX*MAX],qy[MAX*MAX],inc,sf,cx,cy,i,nx,ny,xx,yy;
    inc=sf=1;
    qx[1]=x;
    qy[1]=y;
    a[x][y]=1;
    while(inc<=sf)
    {
        nx=qx[inc];
        ny=qy[inc];
        for(i=1;i<=4;i++)
        {
            xx=nx+dx[i];
            yy=ny+dy[i];
            if(a[xx][yy]==0||a[xx][yy]>a[nx][ny]+1)
```

```
            {
                sf++;
                qx[sf]=xx;
                qy[sf]=yy;
                a[xx][yy]=a[nx][ny]+1;
            }
        }
        inc++;
    }
}
void cautatarm()
{
int i,j,k,ii,jj,v;
for(i=1;i<=n;i++)
    for(j=1;j<=m;j++)
    if(a[i][j]==-1)

    for(k=1;k<=4;k++)
{
    ii=i+dx[k];
    jj=j+dy[k];
    if(a[ii][jj]>=0)
      lee(ii,jj);
}
}
int minim()
{
    int i,j,k,ii,jj,min=MAX*MAX;
    for(i=1;i<=n;i++)
        for(j=1;j<=m;j++)
        if(a[i][j]==-2)
    {
        for(k=1;k<=4;k++)
        {
            ii=i+dx[k];
            jj=j+dy[k];
            if(a[ii][jj]>0&&a[ii][jj]<min)
                min=a[ii][jj];
        }
    }
    return min;
}
int main()
{
    citire();
```

```
    int i;
    numara();
    cautatarm();
    g<<ni[1]<<" "<<ni[2]<<" "<<ni[3]<<" "<<minim()<<endl;

    afisare();
    return 0;
}
```